A Nonlinear Real Arithmetic Fragment[⋆]

Ashish Tiwari and Patrick Lincoln

SRI International, Menlo Park, CA. `firstname.lastname@sri.com`

Abstract We present a new procedure for testing satisfiability (over the reals) of a conjunction of polynomial equations. There are three possible return values for our procedure: it either returns *a model* for the input formula, or it says that the input is *unsatisfiable*, or it fails because the applicability condition for the procedure, called the eigen-condition, is violated. For the class of constraints where the eigen-condition holds, our procedure is a *decision procedure*. We describe satisfiability-preserving transformations that can potentially convert problems into a form where eigen-condition holds. We experimentally evaluate the procedure and discuss applicability.

1 Introduction

Satisfiability problems in nonlinear real arithmetic arise in several applications, including formal verification and synthesis of software programs, control systems, and cyber-physical systems. In this paper, we consider the problem of checking satisfiability of a *conjunction of multilinear polynomial equations* over the reals.

There has been significant progress recently on solving nonlinear real arithmetic constraints [12,13,9,10,16,4,1,7]. Our main interest is identifying efficiently decidable nonlinear arithmetic fragments that arise in formal verification and synthesis, and developing procedures for those fragments that easily integrate with and complement existing techniques in SMT [8]. We present here a procedure that is tailored for a subclass of nonlinear problems that have *finitely-many* (maybe zero) models over an algebraically closed field (complex numbers). Our procedure can be viewed as inspired by SAT solvers: we search for a model by finding the finitely-many values a variable can potentially take in any model, and then nondeterministically guessing the right value. Whereas in SAT, each variable is known *a priori* to take one of two values, in our setting, we have to do some work to determine if there is a variable that takes only finitely-many values. We describe the procedure and report preliminary experimental results.

Why restrict to conjunction of equations? Consider a simple loop that computes the product of two input natural numbers x_0, y_0:

```
s := 0; y := y₀ ; while (y > 0) { s := s + x₀; y := y-1 }
```

⋆ Supported in part by DARPA under contract FA8750-12-C-0284 and by the NSF grant SHF:CSR-1017483. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

Suppose we want to find a loop invariant of the form $s = ax_0y_0 + bx_0y$ (we could pick a general degree 2 polynomial over x_0, y_0, y here, but just to keep expressions small we picked a restricted template here). We want to know if

$$\exists a, b \forall s, x_0, y_0, y, s_1, y_1 : s = ax_0y_0 + bx_0y \wedge s_1 = s + x_0 \wedge y_1 = y - 1 \Rightarrow s_1 = ax_0y_0 + bx_0y_1$$

We can answer the above by checking if the right-hand side polynomial can be written as a sum of (multiples of) the polynomials on the left. Again picking just the minimal template for the multipliers for ease of presentation, we get

$$\begin{aligned}&\exists a, b, u, v, w : \forall s, x_0, y_0, y, s_1, y_1 :\\ &\quad s_1 - ax_0y_0 - bx_0y_1 = u(s - ax_0y_0 - bx_0y) + v(s_1 - s - x_0) + wx_0(y_1 - y + 1)\end{aligned}$$

Equating the coefficients of the monomials over the $\forall$ variables, we get

$$\exists a, b, u, v, w : 1 = v \wedge -a = -ua \wedge -b = w \wedge 0 = u - v \wedge 0 = -ub - w \wedge 0 = -v + w$$

which is a *conjunction of polynomial equations*; see also [15,23,21,27,11,22,25].

A Running Example. We illustrate the main idea in the new procedure for satisfiability testing of nonlinear equations using a small example. Consider the conjunction of the following three equations:

$$x_1x_2 - x_1x_3 = -2x_2 \qquad x_1x_2 = x_3 \qquad x_2x_3 = 1$$

The first two equations can be written in matrix notation as

$$\begin{pmatrix} x_2 - x_3 \\ x_2 \end{pmatrix} x_1 = \begin{pmatrix} -2x_2 \\ x_3 \end{pmatrix}$$

Here, it is possible to write the right-hand side vector, $(-2x_2; x_3)$, as a linear combination of the vector, $(x_2 - x_3; x_2)$, on the left-hand side. Doing so, we get

$$\begin{pmatrix} x_2 - x_3 \\ x_2 \end{pmatrix} x_1 = \begin{pmatrix} 0 & -2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} x_2 - x_3 \\ x_2 \end{pmatrix}$$

This constraint can be true iff either x_1 is an eigenvalue of the 2×2 matrix or the vector $(x_2 - x_3; x_2)$ is identically zero. The two eigenvalues of the matrix are -1 and 2.

Let us branch on the three cases. In the first branch, $x_1 = -1$. The original three equations simplify to $x_2 + x_3 = 0,\ x_2x_3 = 1$. Recursively applying the same analysis, we find that these two equations can be written as

$$\begin{pmatrix} 1 \\ x_3 \end{pmatrix} x_2 = \begin{pmatrix} -x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ x_3 \end{pmatrix}$$

Hence, the value of x_2 should be an eigenvalue of the matrix $(0, -1; 1, 0)$ or the vector $(1; x_3)$ should be identically zero. There are no real eigenvalues of this matrix and the vector $(1; x_3)$ can never be equal to 0. Hence, we get a contradiction in each subcase, and we backtrack.

In the second branch, $x_1 = 2$. The original three equations simplify to $2x_2 = x_3,\ x_2x_3 = 1$. Again, we rewrite the two equations in matrix notation as

$$\begin{pmatrix} 2 \\ x_3 \end{pmatrix} x_2 = \begin{pmatrix} x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0.5 & 0 \end{pmatrix} \begin{pmatrix} 2 \\ x_3 \end{pmatrix}$$

Hence, the value of x_2 should be an eigenvalue of the matrix $(0, 1; 0.5, 0)$ or the vector $(2; x_3)$ should be identically zero. The matrix has two real eigenvalues, namely $\pm\sqrt{2}/2$. If we pick $x_2 = \sqrt{2}/2$, and continue, we find that we find a value $\sqrt{2}$ for x_3 and thus get a model for the original three constraints.

2 Search-based Procedure

In this section, we formally describe our satisfiability checking procedure for nonlinear equations.

We first fix some notation. Let X be a finite set of variables. Elements of X are denoted by x, y with possible subscripts. We use $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ to denote the set of rationals, (algebraic) reals and complex numbers respectively, and we use c, d with subscripts to denote elements of these sets. The set of polynomials over X with coefficients in $\mathbb{R}$ is denoted by $\mathbb{R}[X]$, and its elements are denoted by p, q with possible subscripts. Let us assume that we can represent and compute over algebraic numbers. Our description of the procedure will represent and compute using constants in $\mathbb{R}$, but all these constants will be algebraic.

A (partial) model M is simply a set of assignments $x \mapsto c$ where x is a variable and c is an algebraic real number from $\mathbb{R}$. Each variable x occurs at most once in M.

The input to our procedure is a set $S := \{p_1 = 0, \ldots, p_n = 0\}$ of polynomial equations where every $p_i \in \mathbb{Q}[X]$. The output is either a model M binding every variable occurring in an input polynomial to a constant, or a string "Unsatisfiable" or a string "Condition Failed".

We describe our search procedure using inference rules that operate over the state (S', M') consisting of the set S' of equations, and partial model M'. The initial state is $(S, \emptyset)$, where S is the input equations. The procedure works by applying one of the inference rules in Figure 1. The `Split` inference rule makes a *non-deterministic* guess. Starting from the initial state, if we are able to reach a state $(\emptyset, M)$ using the inference rules, then we output the model M (Rule `Success`). If every derivation starting from $(M, \emptyset)$ (irrespective of the guesses) reaches a contradiction $\perp$, then we output the string "Unsatisfiable". In all other cases, we output the string "Condition Failed".

Among the inference rules in Figure 1, the rules `Fail`, `Delete`, and `Success` are self explanatory. The rule `Unit_Prop` checks to see if there is an equation of the form $x = c$ in S, and if so, it adds it to the model M and replaces x by c in S (the result is denoted by $S[x \mapsto c]$). We assume expressions are normalized to polynomial forms with leading coeffient 1.

The rule `Split` first checks if the set S satisfies the eigen-condition.

`Split`: $$\frac{(S \cup \{A\boldsymbol{v} = x\boldsymbol{v}\}, M)}{(S \cup \{x = \lambda_1\}, M) \ \mid \ \ldots \ \mid \ (S \cup \{x = \lambda_k\}, M) \ \mid \ (S \cup \{\boldsymbol{v} = 0\}, M)}$$
where $\lambda_1, \ldots, \lambda_k$ are all the real eigenvalues of A.

`Unit_Prop`: $$\frac{(S \cup \{x = c\}, M)}{(S[x \mapsto c], M \cup \{x \mapsto c\})}$$ `Success`: $$\frac{(\emptyset, M)}{\text{output model M}}$$

`Delete`: $$\frac{(S \cup \{0 = 0\}, M)}{(S, M)}$$ `Fail`: $$\frac{(S \cup \{1 = 0\}, M)}{\bot}$$

Figure1. Inference rules describing the satisfiability checking procedure.

Definition 1 (eigen-condition). *A set S satisfies the eigen-condition if there exists a variable x such that some subset $S_1 \subseteq S$ of k equations can written in the form $x\boldsymbol{v} = A\boldsymbol{v}$ for some $(k \times 1)$-vector $\boldsymbol{v}$ of polynomials in $\mathbb{R}[X - \{x\}]$ and some constant $(k \times k)$-matrix A in $\mathbb{R}^{(k \times k)}$.*

If the set S satisfies the eigen-condition and $A, x, \boldsymbol{v}$ are the corresponding witnesses, then the inference rule `Split` non-deterministically picks either a *real* eigenvalue of A as the value of x, or sets $\boldsymbol{v}$ to 0. Note that if A has no real eigenvalues, then setting $\boldsymbol{v}$ to 0 is the only option. Figure 2 in the appendix illustrates the inference rules on the running example.

The eigen-condition can be efficiently tested using a greatest fixpoint procedure: we start with a set $T = \{p_i x = q_i \mid p_i, q_i \in \mathbb{R}[X - \{x\}], i = 1, 2, \ldots\}$ containing all polynomials in S that are linear in x. In each iteration, we remove one element, say $p_i x = q_i$, from T if q_i is not in the linear subspace spanned by all the p_j's (i.e., if q_i can not be written as a linear combination of p_j's) in the monomial basis. If the fixpoint is nonempty, the eigen-condition holds for S. If the fixpoint is empty for all choices of x, the eigen-condition is violated for S.

The monomial basis of polynomials and eigenvalues of matrices have been used for discovering (formal power series) invariants for nonlinear (hybrid) systems [17,20]. Our work uses similar ideas, but to more generally find models for nonlinear polynomial equations.

The soundness of the procedure is straight-forward. We state it without proof here.

Theorem 1 (Soundness). *Let $S := \{p_1 = 0, \ldots, p_n = 0\}$ be a set of polynomial equations where each $p_i \in \mathbb{Q}[X]$. Starting from the state $(S, \emptyset)$, if there is a derivation that reaches $(\emptyset, M)$, then M is a model for S in the theory $\mathbb{R}$ of reals. Starting from $(S, \emptyset)$, if every derivation reaches the state $\bot$, then S is unsatisfiable in the theory $\mathbb{R}$ of reals.*

The procedure can fail on certain inputs. We provide some examples below where the procedure fails. These examples will motivate some "pre-processing" steps that will make the procedure "fail" less often, and also lead to a characterization of the class of problems for which the procedure will not fail.

Example 1. Consider the set $S_{\mathtt{quad}} = \{x^2 + 2x + 2 = 0\}$, the set $S_{\mathtt{gb}} = \{x = 2y, x = 3y\}$, and the set $S_{\mathtt{inf}} = \{x = y\}$. None of these sets satisfy the eigen-condition. None of the inference rules is applicable on the state $(S, \emptyset)$, where S is one of the above sets, and hence, our procedure "fails" on each of them.

A polynomial is called *multilinear* if every variable has degree at most one in every monomial in that polynomial. Note that $x^2 + 2x + 2$ is not multilinear, whereas $xy + 2x + 2$ is multilinear.

3 Transformations: Toward Completeness

In this section, we describe satisfiability-preserving transformations, and also characterize the class of problems where our procedure will not fail.

The first transformation, called *multilinear transformation*, turns a non-multilinear polynomial equations (such as $x^2 + 2x + 2 = 0$) into a multilinear equations by introducing new *clone* variables. Instead of defining it formally, we just illustrate it on the example $S_{\mathtt{quad}}$. The multilinear transformation transforms $S_{\mathtt{quad}}$ into the *equi-satisfiable* set $S'_{\mathtt{quad}} = \{x = x_{\mathtt{clone}}, xx_{\mathtt{clone}} + 2x + 2 = 0\}$. The set $S'_{\mathtt{quad}}$, which can be written as $(1; x_{\mathtt{clone}} + 2)x = (-2, 1; -2, 0)(1; x_{\mathtt{clone}} + 2)$, satisfies the eigen-condition and our procedure can detect that it is unsatisfiable. Note that if p is a polynomial over a single variable, then the multilinear transformation transforms $p = 0$ into a equation set $x\boldsymbol{v} = A\boldsymbol{v}$ such that p is the characteristic polynomial of A.

The second transformation, called *Gröbner basis (GB) transformation*, applies the inference rules for computing Gröbner basis [3,2] to the polynomials in S. If these GB computation rules are exhaustively applied, then $\{p_1 = 0, \ldots, p_n = 0\}$ is replaced by the equi-satisfiable $\{q_1 = 0, \ldots, q_k = 0\}$ where $\{q_1, \ldots, q_k\}$ is a Gröbner basis of $\{p_1, \ldots, p_n\}$. Again, we just illustrate the GB transformation by an example. Using the GB transformation, the set $S_{\mathtt{gb}}$ is transformed into the set $S'_{\mathtt{gb}} = \{x = 2y, y = 0\}$, which can be deduced to be satisfiable by our procedure using two applications of the `Unit_Prop` inference rule.

Finally, consider the third example, $S_{\mathtt{inf}} = \{x = y\}$, on which our procedure fails. If our procedure does not fail on a set S, then it implies that the set S has *finitely many* (maybe zero) models. The set $S_{\mathtt{inf}} = \{x = y\}$ has infinitely-many models, and hence our procedure necessarily fails on it; moreover, any simple "model count preserving" transformation will not help.

We will refer to the procedure based on the original inference rules *plus* the inference rules for computing multilinear transformations and Gröbner bases, as the *extended procedure.*

Theorem 2 (Completeness). *Let $\mathcal{S}$ be the class of polynomial equation sets that have finitely-many (maybe zero) models over the complex numbers. If $S \in \mathcal{S}$, then the extended procedure will never fail on S.*

Proof (Sketch). If S is unsatisfiable over the complex numbers, then by Hilbert's Nullstellensatz, the Gröbner basis of S will be $\{1 = 0\}$, which will be detected as "unsatisfiable" by the `Fail` rule. Note that we only need the GB rules and the `Fail` rule in this case. If S is satisfiable and has finitely many models over the complex numbers, let $c_1, \ldots, c_m$ be all the (complex) values that some specific variable, say x, takes in all the models. Consider the polynomial $p := (x-c_1)(x-c_2)\cdots(x-c_m)$ in $\mathbb{C}[x]$. Since S contains polynomials in $\mathbb{Q}[X]$, the polynomial p is in $\mathbb{R}[x]$. If we compute GB using a purely lexicographic ordering where x has the least precedence, then p^k will belong to the GB (for some $k > 0$, by Hilbert's Nullstellensatz). Using the multilinear transformation, the single variable equation, $p^k = 0$, can be transformed into a set of equations where eigen-condition holds and the real values among $c_1, \ldots, c_m$ can be computed using the `Split` rule. The argument is then recursively applied to prove completeness. $\square$

While the main completeness result is likely to be of only theoretical interest, it suggests that we can improve applicability by *lazily* applying GB transformation steps. Our model searching procedure complements the GB procedure that detects unsatisfiability (over complex numbers). In analogy to SAT solving, GB procedure is similar to a resolution-based procedure, whereas our procedure is similar to the DPLL algorithm [6] (for nonlinear arithmetic).

4 Experiments

Boolean SAT problems can be encoded as nonlinear real arithmetic problems: for e.g., the clause $\neg x \vee y \vee z$ can be encoded as $x(1-y)(1-z) = 0$. These nonlinear problems satisfy the eigen-condition and our base procedure never fails on them. It performs DPLL-style search for a model on these examples. In fact, optimizations that have been developed for SAT (conflict-driven clause learning) can be adapted and incorporated into our nonlinear procedure.

Our procedure also works well on problems coming from template-based verification and synthesis [23,15] (see some nonlinear benchmark examples in [14]) and hybrid systems [21,27,11,22,25]. In fact, using a preliminary (and rather naive) implementation of our procedure (in Python, using floats and not using algebraic numbers) with some heuristics for handling cases where eigen-condition fails, we were able to solve all the nonlinear examples in [14] in time competitive with Z3 [13,18] (and faster than Z3 on a couple of problems). On the nonlinear encodings of SAT benchmarks, we are competitive with Z3's nonlinear solver on small problems, but much worse when problem size is larger – this is perhaps because our implementation does not learn from conflicts.

General nonlinear satisfiability problems, as well as $\exists\forall$ nonlinear problems [5], can both be turned into a conjunction of polynomial *equations* using the Positivstellensatz [24,19,26], and our procedure was partly motivated by these problems. For example, $\forall(x, y) : (x \geq 0 \wedge y \geq 0 \Rightarrow xy \geq -x)$ can be turned into

$$\exists(a, b, c, d) : \forall(x, y) : (xy + x = (ax + b)(cy + d) \wedge a \geq 0 \wedge b \geq 0 \wedge c \geq 0 \wedge d \geq 0),$$

which is equivalent to the *conjunction of equations*: $\exists(a, b, c, u, v, w, t) : (ac = 1 \wedge ad = 1 \wedge bc = 0 \wedge bd = 0 \wedge a = u^2 \wedge b = v^2 \wedge c = w^2 \wedge d = t^2)$. If we are careful and introduce the $\exists$ variables so that there are only finitely-many choices (such as, by having $a = 1$), then our procedure can usually solve such problems very quickly. Note that the number of variables becomes large very quickly in template-based approaches, and hence the need for dedicated procedures.

5 Conclusion

We presented a backtracking-based search procedure for checking satisfiability of polynomial equations over the reals, which is complete for a subclass of nonlinear problems that have finitely-many (maybe zero) models over the complex numbers. Our procedure can be viewed as a generalization of DPLL-style SAT solving to nonlinear arithmetic. Preliminary results indicate it is effective on a wide range of (exists-forall) nonlinear real arithmetic problems that arise during analysis and synthesis of programs and cyber-physical systems. In future, we plan to improve it using conflict-driven learning and extensively evaluate it.

References

1. B. Akbarpour and L. C. Paulson. Metitarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reasoning*, 44(3):175–205, 2010.
2. L. Bachmair and H. Ganzinger. Buchberger's algorithm: A constraint-based completion procedure. In *Proc. 1st Intl. Conf. Constraints in Computational Logic*, LNCS 845, pages 285–301. Springer, 1994.
3. B. Buchberger. A critical-pair completion algorithm for finitely generated ideals in rings. In *Proc. Logic and Machines: Decision Problems and Complexity*, volume 171 of *LNCS*, pages 137–161, 1983.
4. C.-H. Cheng, H. Ruess, and N. Shankar. Jbernstein: A validity checker for generalized polynomial constraints. In *Proc. 25th Intl. Conf. Comp. Aided Verif., CAV*, volume LNCS 8044, pages 656–661. Springer, 2013.
5. C.-H. Cheng, N. Shankar, H. Ruess, and S. Bensalem. Efsmt: A logical framework for cyber-physical systems. *CoRR*, abs/1306.3456, 2013.
6. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *CACM*, 5(7):394–397, 1962.
7. M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1(3-4):209–236, 2007.
8. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Proc. 16th Intl. Conf. on Computer Aided Verification, CAV 2004*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
9. S. Gao, J. Avigad, and E. M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *Proc. 6th Intl. Conf. Aut. Reasoning, IJCAR*, LNCS 7364, pages 286–300, 2012.
10. S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *Proc. 24th Intl. Conf. Aut. Ded., CADE*, LNCS 7898, pages 208–214. Springer, 2013.
11. S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Proc. 20th Intl. Conf. on Computer Aided Verification, CAV 2008*, volume 5123 of *LNCS*, pages 190–203. Springer, 2008. July 7-14, 2008, Princeton, NJ.

12. H. Iwane, H. Yanami, and H. Anai. An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for optimization problems. In *Proc. Intl. Workshop on Symb. Numeric Comp., SNC*, pages 168–177. ACM, 2011.
13. D. Jovanovic and L. M. de Moura. Solving non-linear arithmetic. In *Proc. 6th Intl. Conf. Aut. Reasoning, IJCAR*, LNCS 7364, pages 339–354. Springer, 2012.
14. J. Leike. Software and benchmarks for synthesis for polynomial lasso programs, 2014. `http://www.csl.sri.com/~tiwari/softwares/synthesis_for_polynomial_lasso_programs_source.zip`.
15. J. Leike and A. Tiwari. Synthesis for polynomial lasso programs. In *Proc. VMCAI*, LNCS 8318, pages 454–472, 2014.
16. U. Loup, K. Scheibler, F. Corzilius, E. Ábrahám, and B. Becker. A symbiosis of interval constraint propagation and cylindrical algebraic decomposition. In *Proc. 24th Intl. Conf. Aut. Ded., CADE*, LNCS 7898, pages 193–207. Springer, 2013.
17. N. Matringe, A. V. Moura, and R. Rebiha. Generating invariants for non-linear hybrid systems by linear algebraic methods. In *Proc. 17th Intl. Static Analysis Symp., SAS*, volume LNCS 6337, pages 373–389. Springer, 2010.
18. Microsoft Research. *Z3: An efficient SMT solver.* `http://research.microsoft.com/projects/z3/`.
19. P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming Ser. B*, 96(2):293–320, 2003.
20. R. Rebiha, N. Matringe, and A. V. Moura. Transcendental inductive invariants generation for non-linear differential and hybrid systems. In *Proc. Hybrid Syst.: Comp. and Cntrl., HSCC*, pages 25–34. ACM, 2012.
21. S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In R. Alur and G. J. Pappas, editors, *Hybrid Systems: Computation and Control, HSCC 2004*, volume 2993 of *Lecture Notes in Computer Science*, pages 539–554. Springer, 2004.
22. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32(1):25–55, 2008.
23. S. Srivastava, S. Gulwani, and J. S. Foster. Template-based program verification and program synthesis. *STTT*, 15(5-6):497–518, 2013.
24. G. Stengle. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Math. Ann.*, 207, 1974.
25. T. Sturm and A. Tiwari. Verification and synthesis using real quantifier elimination. In *Proc. Intl. Symp. on Symb. and Alg. Comp., ISSAC*, pages 329–336. ACM, 2011.
26. A. Tiwari. An algebraic approach for the unsatisfiability of nonlinear constraints. In *Computer Science Logic, 14th Annual Conf., CSL 2005*, volume 3634 of *LNCS*, pages 248–262. Springer, 2005.
27. A. Tiwari and G. Khanna. Nonlinear Systems: Approximating reach sets. In *Proc. 7th Intl. Workshop on Hybrid Systems: Computation and Control, HSCC 2004*, volume 2993 of *LNCS*, pages 600–614. Springer, Mar. 2004.

A Appendix: Illustrating the inference rules

We illustrate the inference rules on the running example in Figure 2.

$$
\begin{array}{ll}
\texttt{Split:} & \dfrac{(\{x_1x_2 - x_1x_3 + 2x_2 = 0, x_1x_2 - x_3 = 0, x_2x_3 - 1 = 0\}, \emptyset)}{(\{x_1x_2 - x_1x_3 + 2x_2 = 0, x_1x_2 - x_3 = 0, x_2x_3 - 1 = 0, x_1 = 2\}, \emptyset)} \\
\texttt{Unit\_Prop:} & \overline{(\{2x_2 - x_3 = 0, x_2x_3 - 1 = 0\}, \{x_1 \mapsto 2\})} \\
\texttt{Split:} & \overline{(\{2x_2 - x_3 = 0, x_2x_3 - 1 = 0, x_2 = 1/\sqrt{2}\}, \{x_1 \mapsto 2\})} \\
\texttt{Unit\_Prop:} & \overline{(\{2/\sqrt{2} - x_3 = 0\}, \{x_2 \mapsto 1/\sqrt{2}, x_1 \mapsto 2\})} \\
\texttt{Unit\_Prop:} & \overline{(\emptyset, \{x_3 \mapsto \sqrt{2}, x_2 \mapsto 1/\sqrt{2}, x_1 \mapsto 2\})} \\
\texttt{Success:} & \overline{\text{output model } x_3 \mapsto \sqrt{2}, x_2 \mapsto 1/\sqrt{2}, x_1 \mapsto 2}
\end{array}
$$

Figure2. Running example using inference rules.